AD-A103 327 DAVID W TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CE--ETC F/6 9/2
SOFTWARE ENGINEERING - A GUIDELINE.(U)
AUG 81 D J HIBBERT
UNCLASSIFIED DTNSRDC/CMLD-81-22 NL

(12)

CMLD-81-22

AD A103327

DAVID W. TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CENTER

Bethesda, Maryland 20084

Software Engineering - A Guideline

by
Dabney J. Hibbert

APPROVED FOR PUBLIC RELEASE:DISTRIBUTION UNLIMITED

Computation, Mathematics
and Logistics Department

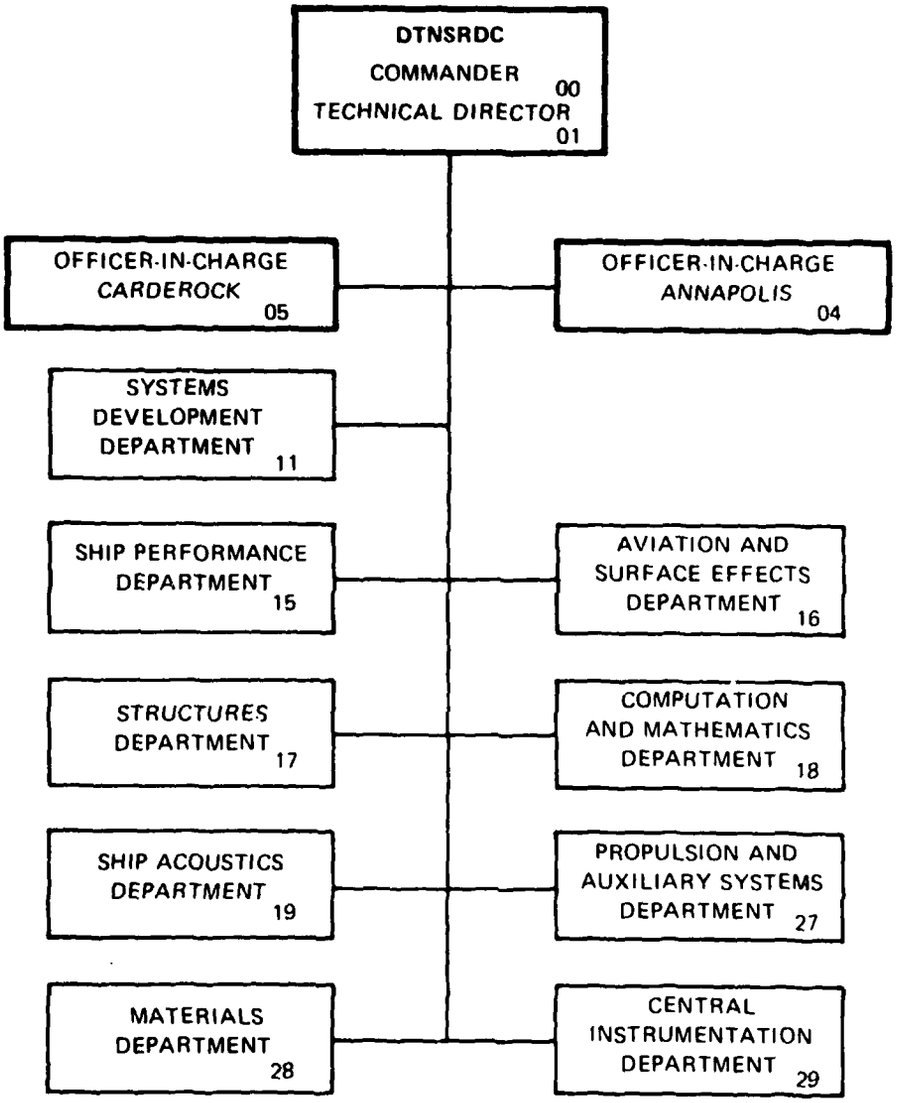
August 1981

CMLD-81-22

Software Engineering - A Guideline

NDW-DTNSRDC 5602/30 (2-80)
(supersedes 3960/46)

MAJOR DTNSRDC ORGANIZATIONAL COMPONENTS



NDW-DTNSRDC 3960/43 (Rev. 11-75)
GPO 905-879

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CMLD-81-22	2. GOVT ACCESSION NO. AD-A103 327	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Software Engineering - A Guideline		5. TYPE OF REPORT & PERIOD COVERED Final 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Dabney J. Hibbert		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS David W. Taylor Naval Ship R&D Center ADP Software Specialist (Code 189.1) Bethesda, Md. 20084		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE August 1981 13. NUMBER OF PAGES 14
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software Engineering, Software Development, Documentation, Structured Programming, Validation, Software Tools

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report gives users of the DTNSRDC computers an overview of the principles of software engineering and describes various software engineering techniques that users with either large or small applications would find beneficial.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

CONTENTS

LIST OF FIGURES

ABSTRACT

This report gives users of the DTNSRDC computers an overview of the principles of software engineering and describes various software engineering techniques that users with either large or small applications would find beneficial.

BACKGROUND & DEFINITION

The term "Software Engineering" was coined during the late 1960's at a NATO Science Committee conference. Since then, many articles and books have been published in an attempt to define this approach to producing and managing software. These definitions are based on the types of theoretical foundations and practical disciplines found in the more common branches of engineering.

> "Software engineering is a discipline, rather than a technology. While it uses technology, it also encompasses the methods and practices of people who produce software. Many of the problems with today's software, viz. its high cost, long delivery time, and unreliability, have been attributed to inadequate software engineering".[1]

> "Software engineering is not just a collection of tools and techniques, it is ... a full-fledged engineering discipline...".[2]

> "...the establishment and use of sound engineering principles (methods) in order to obtain economically software that is reliable and works on real machines."[3]

These authors and others have recognized the basic need of any group responsible for producing or managing software to establish a discipline, technique, or method by which reliable, flexible, and timely software is developed, tested, and maintained.

PURPOSE

The decrease in hardware costs and increase in software costs in recent years, combined with the growing lag between computer hardware development and software technology, have produced what many experts have called the first part of a "software crisis". This continuing trend is reflected in Figure 1.

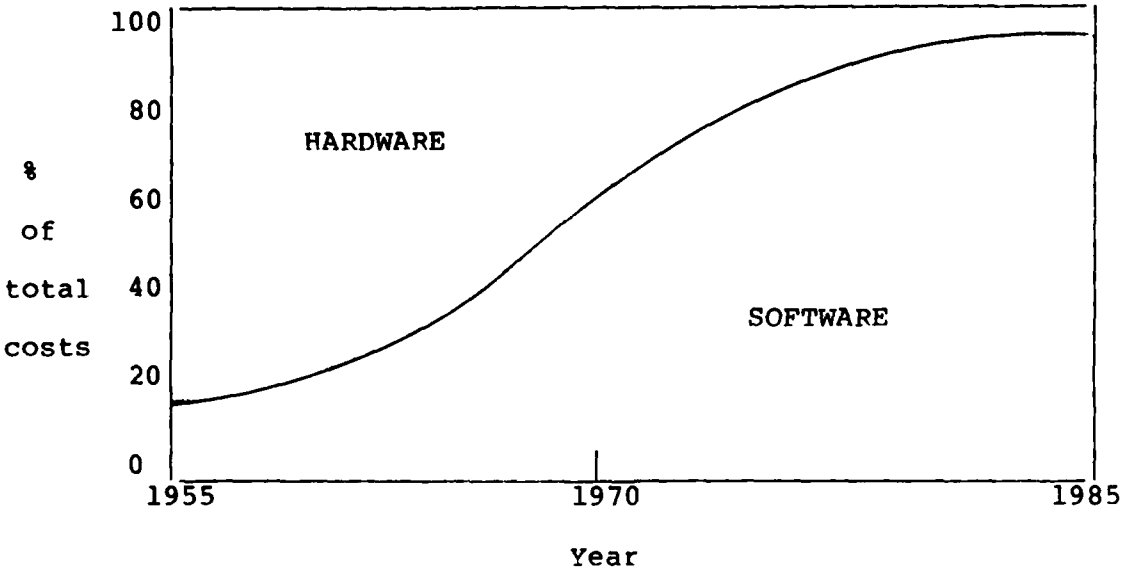


Figure 1. Hardware/Software Cost Trends
(Source of Data; Boehm [4])

As the chart illustrates, during the last 25 years, both the cost of raw computing power and the per unit cost of memory and storage have been dramatically reduced. At the same time, the cost of producing the additional software necessary to take advantage of this increasing computer capacity continues to rise.

Deficiencies in the areas of talent and effective tools/methods make up the second part of the "software crisis". There is a continuing shortage of talent in the software areas of design, development, and management. This skilled manpower shortage has often caused projects to be carried out by a mix of people with differing types of data processing skills and experience. Also, few software projects have been completed within the constraints of original cost, schedule, and performance specifications. A large part of the cost overrun problem is due to a growing lag in the development of software tools and methods to manage the growing complexity of sophisticated software systems. Lack of planning and hardware problems also contribute to cost overruns.

Additionally, when software evolves in a continually changing R&D environment, it is especially important to attempt to "extend code life expectancy by stressing designs with built-in flexibility and re-utilization potentials."[5]

Software management can be defined as "the technical and management control throughout the life cycle of all software that supports an organization's mission and objectives."[6] This management is the responsibility of the computer user organization. It serves to establish accountability for project decisions and objectives and also provides upper management with visable measures of progress toward desired goals. Good software management, initiated at the beginning of a project's life cycle, tends to produce a higher quality software product. High-quality software can be defined as having the following characteristics:

1. Reliability
2. Clarity
3. Maintainability
4. Portability
5. Efficiency/Economy
6. Testability

Many authorities agree that a software engineering approach will alleviate many software development problems.

> "An engineering effort involves a specific sequence of steps beginning with a general plan, proceeding through detailed design, and concluding with implementation of the design."[7]

When applied to a software design effort, it assures the likelihood that the final product will better resemble the system design, while allowing individual creativity and innovation.

CONCEPTS AND SUGGESTED TECHNIQUES

The following section describes some software engineering techniques that may be used to achieve the aforementioned goals of high-quality, economical, and timely software in the areas of software development

PROJECT DEVELOPMENT TEAM APPROACH

A project development team is often the basic organization used to improve manageability and productivity of software projects. This approach allows management to assemble the necessary

mixture of various skills from the talent available and is an effective method for overcoming the talent shortage. Additionally, the project team approach allows for peer review, better dissemination of technical and status information, and better back-up of knowledgeable personnel. This approach also allows for a certain amount of on-the-job training when a team consists of both trainee and journeyman personnel. The optimum design group is made up of two-man teams, each working on a module or project element. Figure 2 illustrates a suggested organizational structure for a large project.

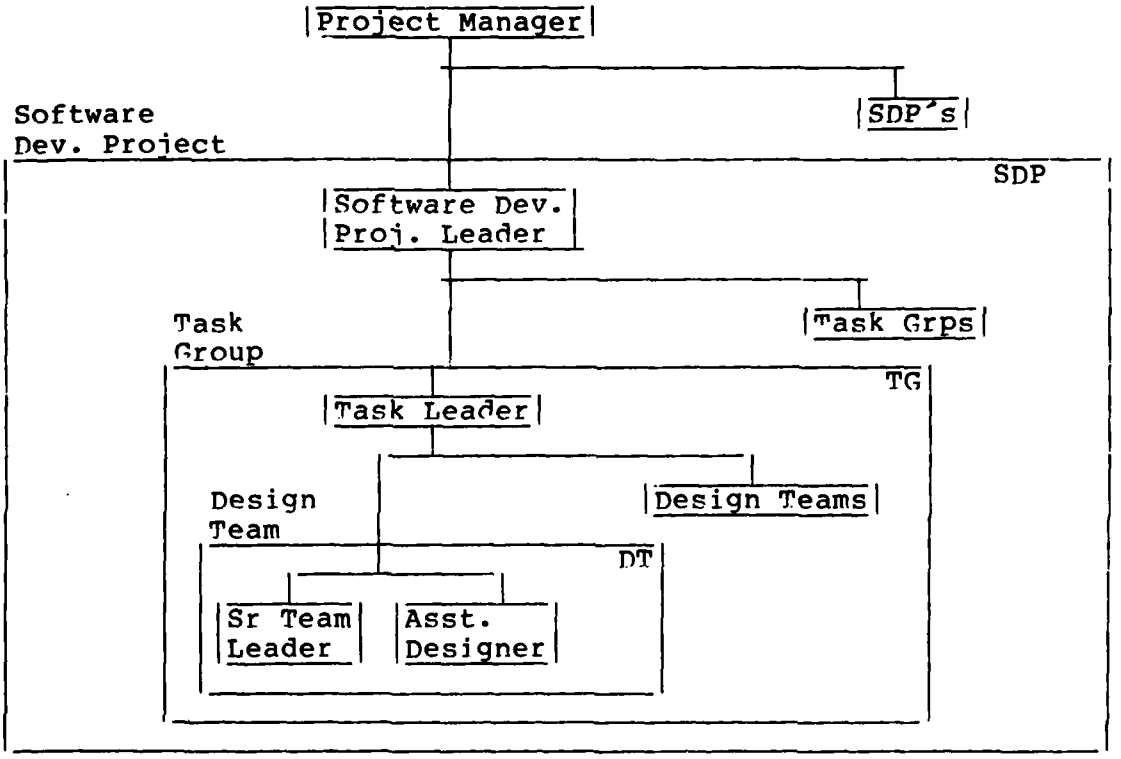


Figure 2. Project Development Group Organization
(Source of Data; Jensen [2])

At each level, the leader is technically involved. The short communication paths ensure good communication between the project team members. Periodic design walkthroughs with the other elements ensure correct interfacing. This structure may be scaled-down for smaller projects, eventually to as small as two people, with the more experienced person being the project leader as well as part of the design team.

The advantages of this approach are:

- the more effective use of human resources

- a better design due to continuous walkthroughs

- insurance against loss of personnel

- an improved training atmosphere for "junior" programmers

DOCUMENTATION

Documentation is a key element at all levels of project development. System requirement documents and technical reviews serve as quality controls for the Definition and Initiation phases of a project. An on-going Project Record, initiated at project conception, should continue during the system life. The Project Record is an open, official record of the major decisions and pertinent documents and specifications. Note that the Project Record is not a Technical Notebook, which is also an effective control method used as an informal medium for exchanging tentative proposals among designers and programmers. Design Specifications, specifying the input and output of each process and the data flow interconnections, and a formal Test Plan, detailing the testing schedule and range of test cases, are two other recommended documents.

STRUCTURED PROGRAMMING

Structured programming techniques, a concept that encompasses programming team management, design methods, and programming technology, may also be used. Of the many definitions of this basic, scientific methodology cited in software literature, perhaps the most preferred is:

> ""Structured programming" is the formulation of programs as hierarchical, nested structures of statements and objects of computation."[8]

Structured programming is often referred to as "modular" or "top-down" programming. The many methods and programming techniques vary widely, but all have as their objective a more organized and more manageable software product achieved by subdividing a complex problem into its basic elements. Some structured programming methodologies are listed in Figure 3. Complete descriptions of the individual approaches can be found in the references.

This partitioning technique allows the creation of well-defined, functional modules and focuses attention on interfaces among modular entities. Another important benefit is the identification of distinct, independant software development tasks. The project leader is thereby able to assign design responsibility for a new process or procedure to a team, which "proceeds in

NAME	APPROACH	APPLICATIONS
Top-Down Design [9] [10]	Based upon decomposition by trial and error	Data processing and and algorithmic*
Structured Design [11]	Based upon decomposition determined by data flow or data transformations	Data processing and and algorithmic*
Jackson Method [12]	Based upon decomposition determined by data structure	Data processing

*Implementation determined primarily by algorithm that defined function (e.g. Gauss-Seidel iterative method for solving system of linear equations)

Figure 3. Structured Programming Methodologies (Source of Data; Jensen [2])

a top-down fashion to analyze the requirements and formulate a solution."[5]

VALIDATION

Validation, or verification, confirms that a project/system is reliable and meets its specifications as well as the requirements of the user environment. Ideally, an informal design review is held with the tasked project group before implementation begins. Iterative reviews should continue throughout the project life cycle up to the delivery and operation phases.

Additionally, before the design phase is complete, a formal test plan for the system is advisable. This test plan should be jointly agreed upon by the user and design groups. Specific testing criteria and guidelines should be adopted for all programs of the system and the testing should assure a high measure of reliability that would be expected in operation of computer programs. Furthermore, a shakedown period is advisable after the delivered software is installed.

SOFTWARE TOOLS

A standard library of specialized software tools should be established. Software tools are computer programs that automate tasks in the management, production, and testing of software. These tools are usually inexpensive, are often available from other users for the cost of reproduction, and are often concise

and easily modified for unique project needs. Some standard tools are an editor, program analyzer, and program librarian. Text editors are an excellent example of available and useful software tools that serve a standardized, special function sufficiently well and interface readily with other tools/programs. Program analyzers are software which scan a source program and collect data about its execution behavior. Program librarians provide for the storage and retrieval of different versions of program modules

EVALUATION METHODS

Engineering analysis, modeling or simulation, and experience data can be used to establish the performance needed in critical software operations. Concrete, measurable milestones of progress for the design, programming, testing, installation, and initial operation of the system, should also be developed to ensure that effective control of the emerging system can be maintained throughout its life cycle.

WHAT TYPES OF USERS NEED SOFTWARE ENGINEERING?

The following sections are intended to serve as guides to the users of the different computer systems available at DTNSRDC.

SMALL/ONE-TIME APPLICATIONS

Users with small and one-time applications, either on a strictly interactive system or on a combination of interactive and batch, should make use of at least two or three of the Software Engineering techniques described above.

The first technique, documentation, provides an invaluable permanent record of a programmer/scientist's work. Often a small task is later added to and becomes a major effort, or an intended one-time application is used more than once. A complete, permanent description of the initial effort will make the later task easier and more time-efficient. Likewise, it is helpful to have someone not familiar with the project read your documentation and try to follow the instructions provided for data creation and format, input, etc. Any misdirection discovered at this stage will save later users many hours of wasted effort.

The second technique that will be beneficial for small/one-time applications is verification. Often the reactions of a program/system to test data and to real data are radically different. Therefore, it is beneficial to acquire and test with

live data before the final turnover process is initiated.

A third useful technique is the use of software tools, especially text editors which will save time and effort, especially in program modification.

LARGER APPLICATIONS

All of the Software Engineering techiques discussed earlier should be used in larger program/systems with the benefits of time and cost efficiency which grow in proportion to the size of the project.

The benefits that small users derive from documentation and verification are even greater for large programs/systems. In larger projects, where the project development team concept is employed, documentation is a shared on-going responsibility. As separate entities are joined and enhanced, documentation can serve as the coordinating factor. Here too, a final reading by a third party should be included.

Verification of a large project can be less complicated by the use of real test data. Only in this manner can the designers be assured that all program interfaces match and that all interactions are correctly taking place. Correct interfaces and interactions can be further ensured by strict control to guarantee uniform program alterations by programmers.

In large projects, the delegation of specific work assignments is made easier by the Structured Programming technique. The use of this technique, in which a complicated task is subdivided into basic, assignable tasks, makes the project's development more manageable.

As in smaller projects, software tools, such as text editors, can make the program development task easier, faster, and more efficient.

Finally, a written and formally agreed upon test plan, using live data, helps ensure the success of the project's development.

REFERENCES

[1] Software Engineering:Problems and Future Development, J.A. Clapp, Mitre Corp., Prepared for Electronic Systems Division, Nov., 1974.

[2] Jensen, R. W. and C. C. Tonies, Software Engineering, Prentice-Hall, Inc. (1979).

[3] Bauer, F. L., "Software Engineering," Information Processing 71, Amsterdam:North Holland Publishing Co., p. 530, 1972.

[4] Boehm, B. W., "Software and Its Impact:A Quantitative Assessment," Datamation, Vol. 19, No. 5, pp.48-59, (May,1973).

[5] Smith, R. J. II, Development of Design Automation Codes using Software Engineering Methods, Lawrence Livermore Laboratory, Prepared for submission to the Ninth Annual Aislomar Conference on Circuits, Systems and Computers, (Nov 3-5, 1975).

[6] Computer Software Management:A Primer for Project Management and Quality Control, NBS Special Publication 500-11, (July, 1977).

[7] Software Engineering Techniques, Wasserman, A. I., Infotech International, (1977).

[8] Wirth, N., "On the Composition of Well-Structured Programs", Computing Surveys, Vol. 6, No. 4, pp. 247-259, (Dec., 1974).

[9] Mills, H. D., "Top-Down Programming in Large Systems", Debugging Techniques in Large Systems, Prentice-Hall, Inc., pp. 41-55, (1971).

[10] Maynard, J., Modular Programming, Auerbach Publications, (1972).

[11] Stevens, W. P. et al., "Structured Design", IBM Systems Journal, Vol. 13, No. 2, pp. 115-139, (May, 1974).

[12] Jackson, M. A., Principles of Program Design, Academic Press, Inc., (1975).

INITIAL DISTRIBUTION

COPIES:

12 DIRECTOR
DEFENSE DOCUMENTATION CENTER (TIMA)
CAMERON STATION
ALEXANDRIA, VIRGINIA 23314

CENTER DISTRIBUTION

COPIES:

1	18/1809	GLEISSNER, G. H.
1	1804	AVRUNIN, L.
1	1805	CUTHILL, E. H.
2	1809.3	HARRIS, D.
1	182	CAMARA, A. W.
1	184	SCHOT, J. W.
1	185	CORIN, T.
1	187	ZUBKOFF, M. J.
1	189	GRAY, G. R.
60	189.1	HIBBERT, D. J.
1	189.2	HAYDEN, H. P.
1	189.3	COOPER, A. E.
10	1892.1	STRICKLAND, J. D.
1	1892.2	SOMMER, D. V.
1	1892.3	MINOR, L. R.
1	1894	SEALS, W.
1	1896	GLOVER, A.
1	1896.2	DENNIS, L.
1	522	LIBRARY, CARDEROCK
1	522.2	LIBRARY, ANNAPOLIS

DTNSRDC ISSUES THREE TYPES OF REPORTS

1. DTNSRDC REPORTS, A FORMAL SERIES, CONTAIN INFORMATION OF PERMANENT TECHNICAL VALUE. THEY CARRY A CONSECUTIVE NUMERICAL IDENTIFICATION REGARDLESS OF THEIR CLASSIFICATION OR THE ORIGINATING DEPARTMENT.

2. DEPARTMENTAL REPORTS, A SEMIFORMAL SERIES, CONTAIN INFORMATION OF A PRELIMINARY, TEMPORARY, OR PROPRIETARY NATURE OR OF LIMITED INTEREST OR SIGNIFICANCE. THEY CARRY A DEPARTMENTAL ALPHANUMERICAL IDENTIFICATION.

3. TECHNICAL MEMORANDA, AN INFORMAL SERIES, CONTAIN TECHNICAL DOCUMENTATION OF LIMITED USE AND INTEREST. THEY ARE PRIMARILY WORKING PAPERS INTENDED FOR INTERNAL USE. THEY CARRY AN IDENTIFYING NUMBER WHICH INDICATES THEIR TYPE AND THE NUMERICAL CODE OF THE ORIGINATING DEPARTMENT. ANY DISTRIBUTION OUTSIDE DTNSRDC MUST BE APPROVED BY THE HEAD OF THE ORIGINATING DEPARTMENT ON A CASE-BY-CASE BASIS.


DATE
TIME